

What R We Doing?

A beginners guide to using R

Katrina Connell

Dec 2018

Introduction

This manual is intended for someone who is just starting out with R. There are a lot of resources for learning R out there, but I have found that there aren't as many resources for truly starting at 0, and many assume some prior knowledge. This manual was written to bring you from 0 to a starting point of being able to use those resources or to give you a head start in your classes. This manual should have come in a folder of files that you will use later. I hope this proves useful to you!

Contents

0	Before you get started...	2
0.1	File organization	2
0.2	Downloading R	2
0.3	Downloading R Studio	3
0.4	Free Online Tutorial	3
0.5	Getting started with R	4
1	Using R and R Studio	5
1.1	Windows in R Studio	5
1.2	Basic R	6
2	Saving Your Work	7
2.1	Using R Scripts	7
2.2	The R Studio Workspace	10
3	Using R Markdown Scripts	13
3.1	Installing R Markdown	13
3.2	Using R Markdown	15
3.2.1	Tidyverse on Linux	16
3.2.2	Useful Shortcut	16
4	Knitting Markdowns	18
5	R Projects	19
5.1	Directories and File Paths	19
5.2	Using R Projects	21

0: Before you get started...

0.1 File organization

This manual should have come inside a folder with the following structure of files and folders. Please make sure you have this structure starting out. For any analysis you do in the future, I recommend a similar structure with a MAIN FOLDER containing your scripts, and then sub folders for data and graphs.

```
Beginning_R_Manual
Beginning_R_Manual.pdf
Beginning_R_Files
  data
  graphs
```

You will keep your data files in the “data” folder, and any graphs you export will save to the “graphs” folder

Now, you will need to make sure you download R and R Studio.

0.2 Downloading R

Click on the following URL or type it into your browser to navigate to the page where you can download R <https://cran.r-project.org/mirrors.html> Scroll down the list and click on any site in the US or which every site is closest to you.

- Choose your system
- Choose your OS

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

R for Windows

[R-3.5.2.pkg](#)

MD5-hash: 0e4ee0980c9a1799329e0d52445d563e
SHA1-hash: cf1db060d68b6c345ac968892a8626b466a2e3b7
(ca. 74MB)

Latest release:

R 3.5.2 binary for OS X 10.11 (El Capitan) and higher, signed
Tel/Tk 8.6.6 X11 libraries and Tinfo 5.2. The latter two
only needed if you want to use the `rc1tk` R package or build

Binaries for base distribution. This is what you want to [install R for the first time](#).
Binaries of contributed CRAN packages (for R >= 2.13.x; managed by Uwe Ligges).
services and corresponding environment and make variables.

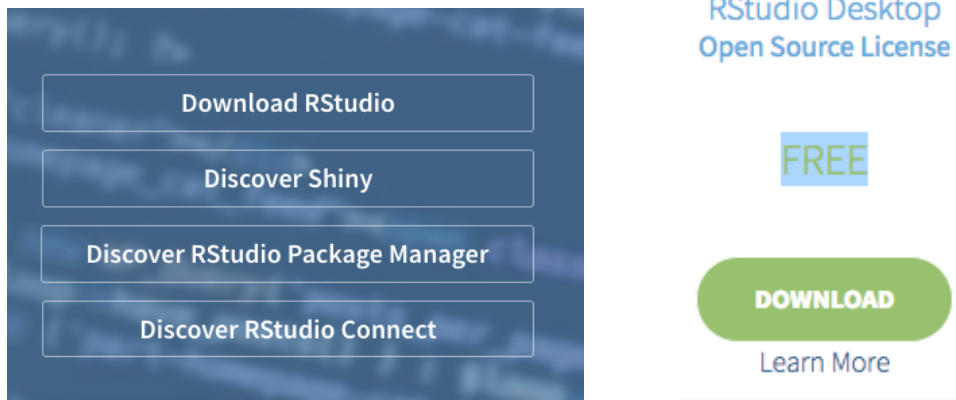
Mac

Windows

0.3 Downloading R Studio

Click on the following URL or type it into your browser to navigate to the page where you can download R Studio <https://www.rstudio.com/products/rstudio/download/>

- Free download
- Choose system



Installers for Supported Platforms
Installers
RStudio 1.1.463 - Windows Vista/7/8/10
RStudio 1.1.463 - Mac OS X 10.6+ (64-bit)

0.4 Free Online Tutorial

If you would like to get a little hands on R practice before we go any further, there is a nice free course offered by DataCamp at the following link. Feel free to click the link below and check it out. <https://www.datacamp.com/courses/free-introduction-to-r>

0.5 Getting started with R

If everything installed correctly, you should now be able to find the R Studio icon (see below) and open it.



When you open R Studio, you should see a screen like that in Figure 1.

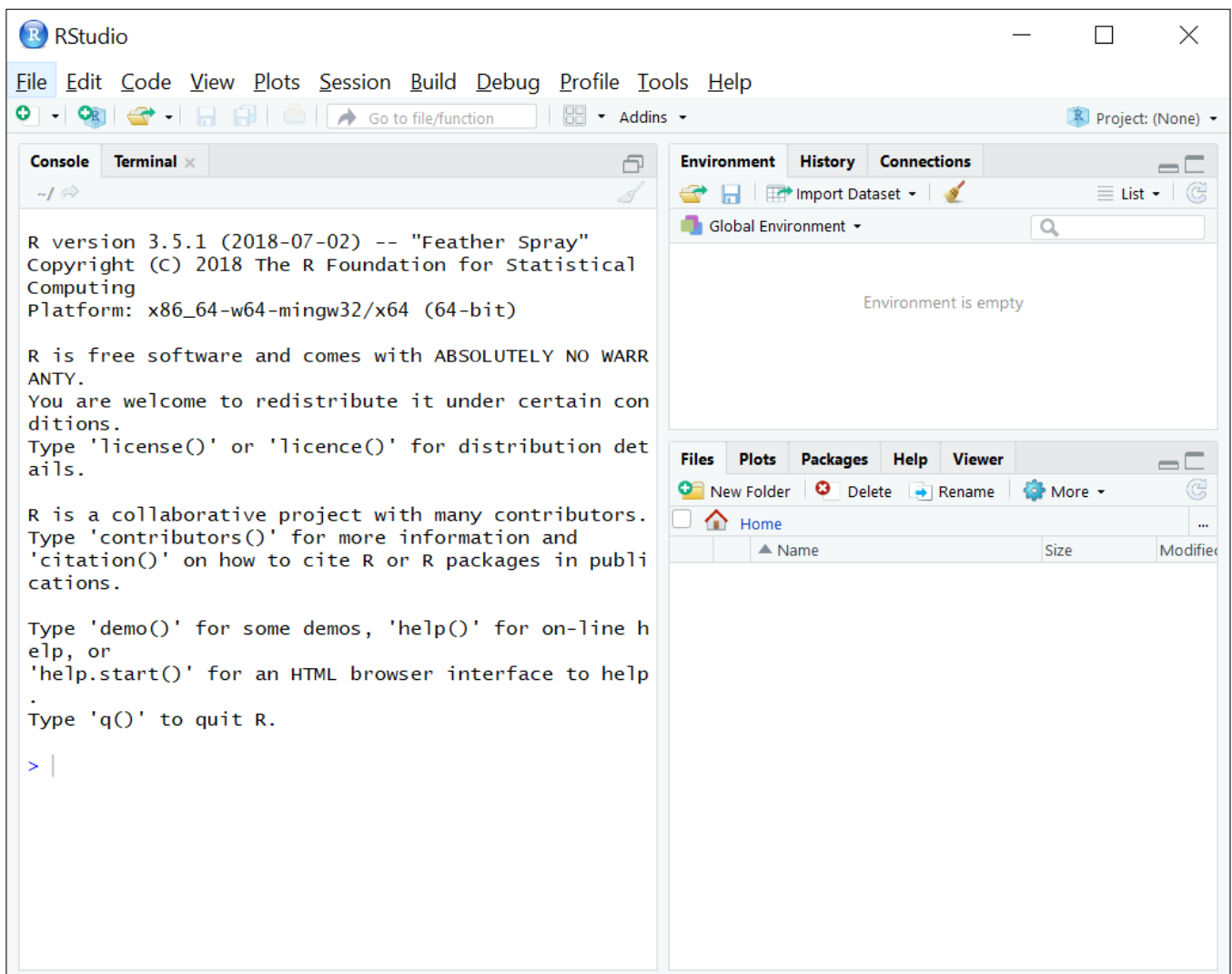


Figure 1: R Studio Opening Screen

1: Using R and R Studio

1.1 Windows in R Studio

Now is a good time to give some names to the different windows in R Studio so we can reference them.

Source Code - The source code window will house whatever file type you are saving your code to. For our purposes right now, this is a script file. This is the only part of your analysis that is "real" or permanent (the "long-term memory").

Console - The console is actually the interface to R. Whatever you type here will be run, but will not be saved. Anything you want saved long term should go into the Source Code. This is more or less the really fancy calculator, and where all of the computations/work is done.

Environment/History Tabs - We will come back to these tabs later, but this is where variables and data sets will be stored (Environment), and where you can see a history of all of the things types into the Console. I will show you later show to save things from the History tab to the Source.

Files/Packages/Plots Tabs - In these tabs you will find a nice interface for managing the packages in R, for loading files and seeing your plots as you make them.

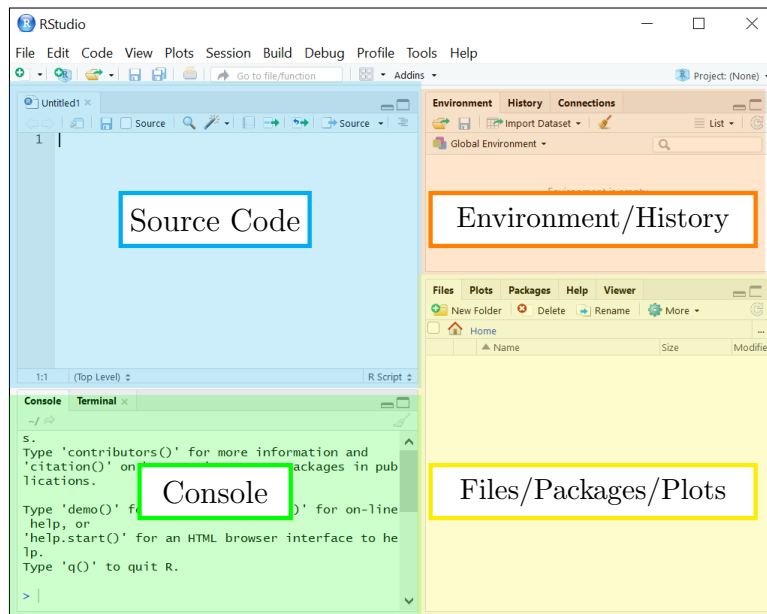


Figure 1.1: Windows in R Studio

1.2 Basic R

In the Console, you will see a `>` and a cursor. Here is where you can type. The most basic thing the R can do is act as a calculator. So, you can type `2+2` and hit ENTER, and it will calculate the results for you.

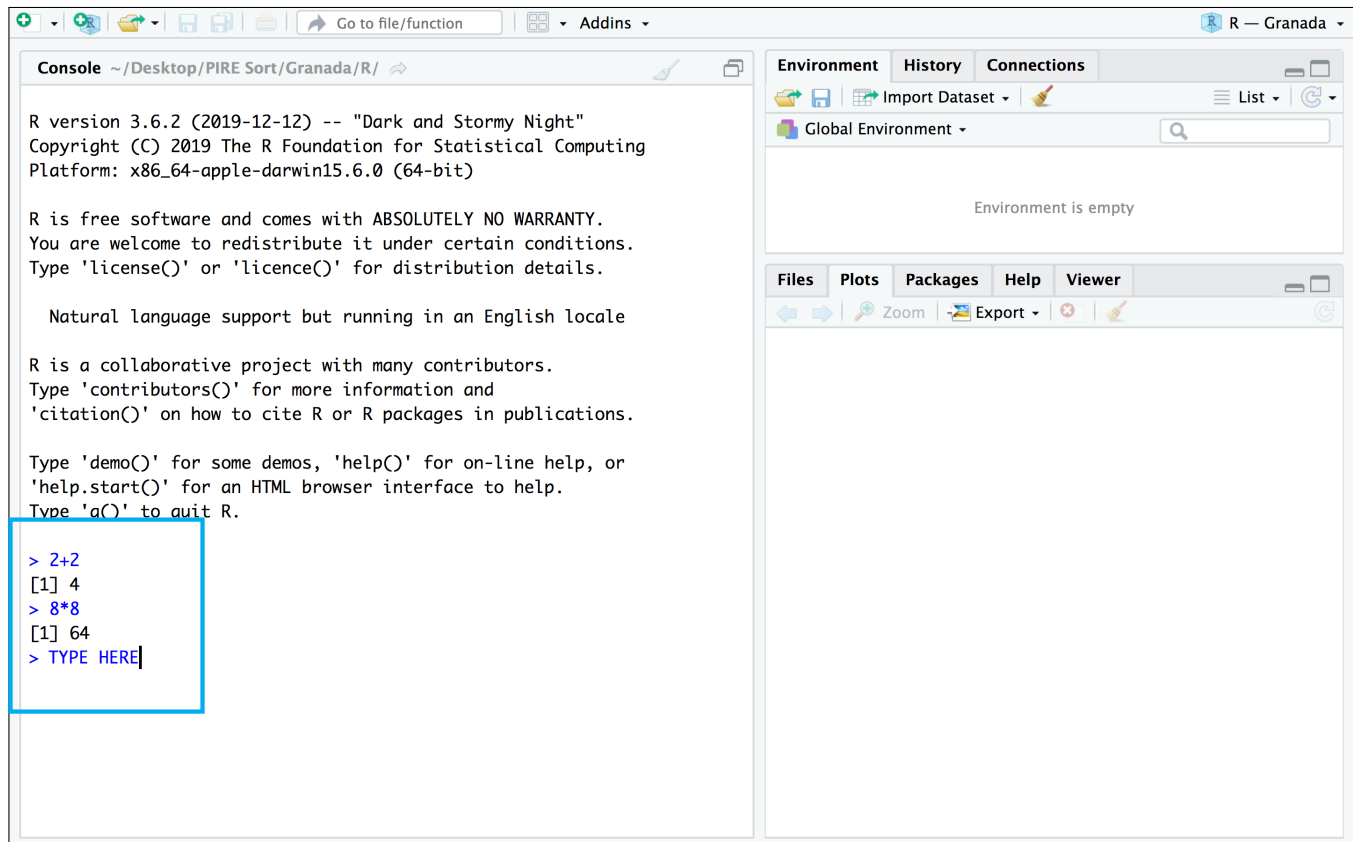


Figure 1.2: R as a Calculator

This is a wonderful feature of R, but once we type it in and run it, it is done. We can't save what we did and run it again later. Imagine you spent hours on an analysis and weren't able to save it to pick it up again later. Whatever we type in the Console is temporary, basically like working memory, but we want to save our work into something more permanent.

2: Saving Your Work

⚠ Very Important!

Code typed into the console is fleeting. You type it, it runs, and it's gone. R will save a history of what type, but this is not useful, and depends on R's memory instead of your computer's memory. You should treat everything that happens in R as temporary except for 2 things: Data files you have saved to your hard drive, and scripts you write and save to your hard drive. So, in this chapter, I will tell you first, *how* you need to save our work, and then I will show you *why*.

2.1 Using R Scripts

One way to save our work is to use R Scripts. Go to FILE->NEW FILE ->R SCRIPT

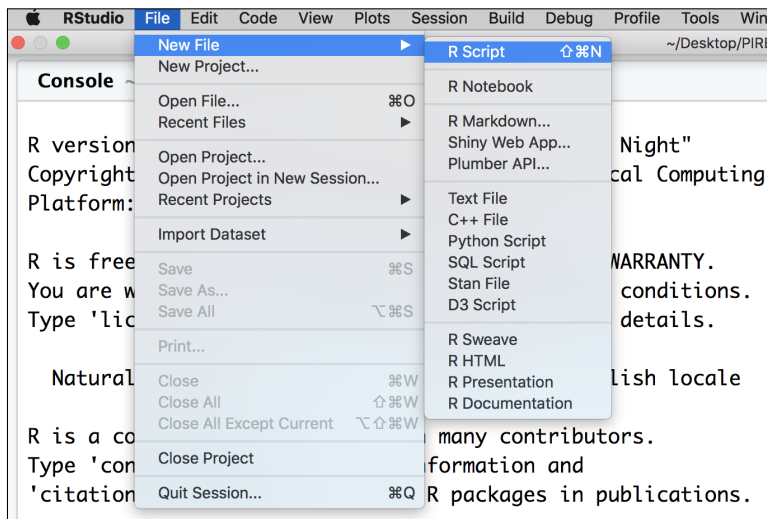


Figure 2.1: Opening an R Script

This will open up a new window (if it didn't automatically open) in the top left window. This window is a text file that you can save code to. So if you now type 2+2 and CMD+ ENTER here, it will simply type this in the text window, run the line of code, and move to the next line. Go ahead and run that line of code and then save that file to the folder you created earlier and name it Rscript.

ⓘ Mac vs. PC Shortcuts

R will use many shortcuts to make working in the environment faster and more efficient. For Mac the commands will always use COMMAND + Something. PC will always use CONTROL + Something. I will use **CMD ENTER** often, which will stand for either (Mac) COMMAND RETURN and (PC) CONTROL ENTER

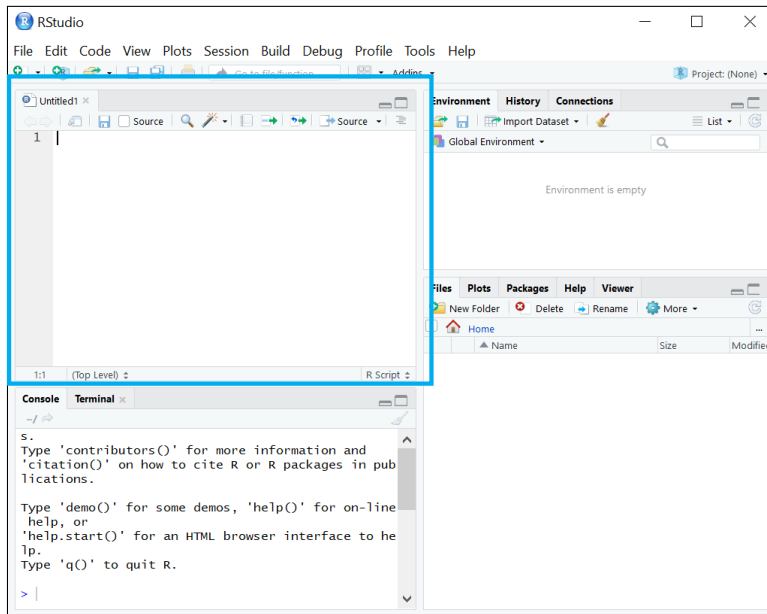


Figure 2.2: R Script

This manual should have come with a folder of files. Make sure you have that whole folder downloaded somewhere you know where to find it (somewhere that is not “Downloads”). Go into that folder and open up the Beginning_R_Script.R file.

In this file, you will see several lines of code . Lines beginning with # are “comments” and will not be treated by R as code. You can use these lines of code to explain what each line of code does so that you or future researchers can understand how the code works.

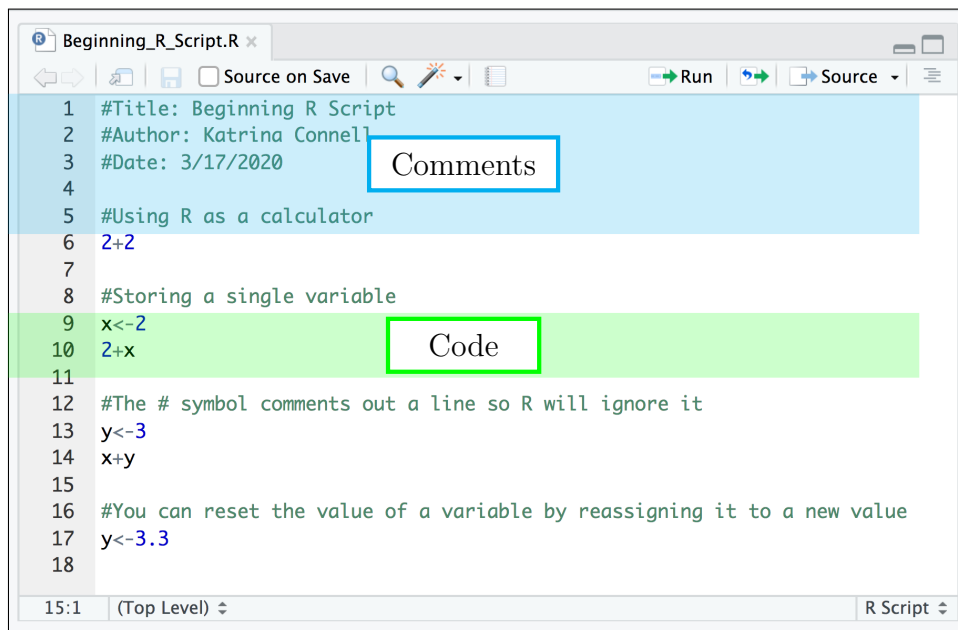



Figure 2.3: Comments and Code

The line that says 2+2 is actual R code, because it has not symbol before it. To run a line of code from the Source, you can do one of three things. I recommend the third option, as it is the most efficient.

1. Highlight the entire line of code and click  at the top left of the Source window.
2. Highlight the entire line of code and press CMD ENTER
3. **Simply click anywhere in the line of code and press CMD ENTER**

Go ahead and try this on the 2+2 line and then look down into the Console to see the result. You should now see the following in the console.

```
#Using R as a calculator  
>2+2  
[1] 4
```

Although the Console shows the comments, it ignored them, and didn't try to evaluate it as code. If you took out the #, you would get an error, because R wouldn't know what to do with just the text. You can think of it as English and R code are 2 different languages. R's job is to translate the R language, so when it sees something in English, it doesn't know what to do with it. But, if we comment it out, R knows not to worry about that part, and will just skip over it. This isn't to say that plain text can't be useful, it can. We just have to tell R what to do with it. We can do this by setting variables. So let's say we have a variable that we will want to use a lot. We can tell R to treat text as that variable. So on the next line down you see...

```
#Storing a single variable  
x<-2  
2+x
```

Go ahead and run both lines and see what shows up in the Console. What you have just done is told R to treat the letter x as the number 2. So when you try to add 2 to x, it gives you 4. Also note that we now have something stored in our Environment Tab. This should now list a value of x and 2, indicating the x is set to the value of 2. When you have longer analyses this will come in handy so you know what variables you have already used.

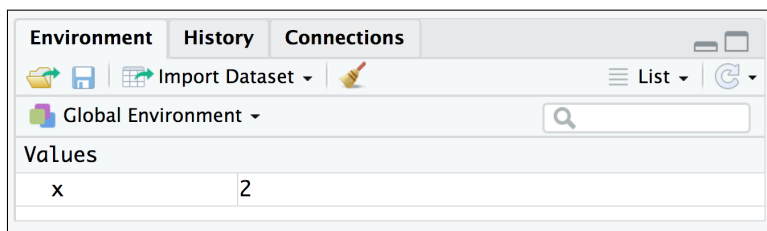


Figure 2.4: Environment Tab

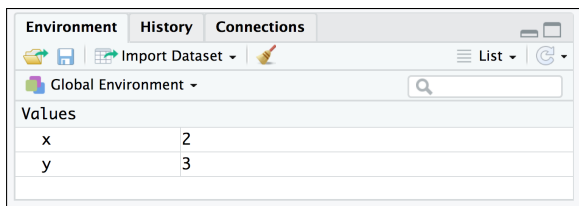


Figure 2.5: y->3

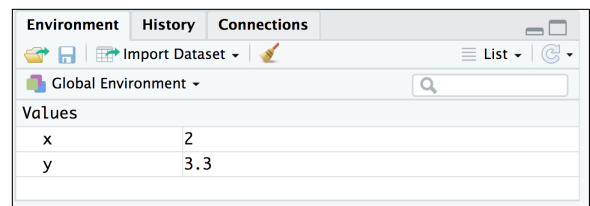


Figure 2.6: y->3.3

You can set another variable `y`, and then add `x` and `y` together (line 12). Something you need to be careful of though is if you REASSIGN the variable to a different value, it will overwrite the old value. This is why you need to keep track of the variables you have used, so you don't accidentally overwrite one by accident.

This is all I am going to teach you with R Scripts. They are a wonderful tool to use, and I know many amazing researchers who use scripts and only scripts. I would like to teach you what I deem to be a more useful tool for creating, managing, and Sharing analyses called the R Markdown file. Everything you have learned so far about scripts apply to Markdowns as well, but there are additional features to the Markdowns that I like and find useful. Before we move onto that though, there is an important discussion we need to have about how we save our work.

2.2 The R Studio Workspace

Let's say, you have finished working on your analysis, and you want to close R Studio. When you close R Studio, it will ask you the following.

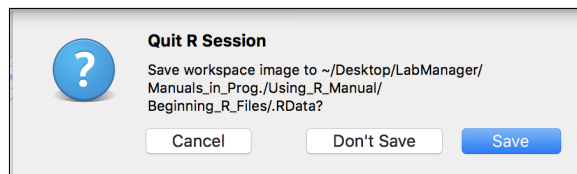


Figure 2.7: Quit R Session?

You may be tempted to say yes, because you just want to be safe, right? Every seasoned R user will tell you “no”. Do not save your R WORKSPACE. Here is why. You have just run the lines of the R script, but now, I want you to go to the console and type the line below and run it.

```
z <-45
```

Now, try to close out R and when it asks you to save the Session, click “yes”. Now, go back and re-open that script. Navigate to the script on your computer and do “OPEN WITH” R studio. You will see that you script opens, and if you look at the history tab, R has saved that you typed and ran the line assigning `z` to 45.

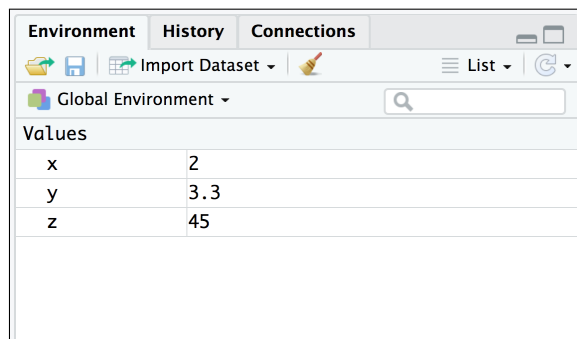


Figure 2.8: Saved R Session Variables

This may look like exactly what you want, but this is a bad habit to get into. R saves your current variables and data frames to a file called “.RData”. When you re-open R **from that working directory**, the workspace will be loaded in, and all will be as it was before. There are a few reasons why this is **not** a good way to go about your analysis. First, this file is easily overwritten, and is often done accidentally. If this happens, anything that was not *saved* in a script is gone for good. This is NOT a safe way to store your analysis. If this history file is overwritten, you will lose that *z* variable entirely, and you will only have what was saved in your actual script file. Imagine setting a variable early on in your analysis, and losing it because it was only saved in the history, not the script? Would you be able to recover it? Maybe. But maybe not. It is better to not risk it. Secondly, in the age of reproducible research, you want to be able to send your analysis to someone. This will mean sending them your data and script, so they won’t have anything that isn’t saved into the script, and will be missing parts of your analysis. So, **EVERYTHING** must be in the script. Treat the R workspace as an incorporeal “spirit realm” of R. It may look like it’s always there, but when you go to look for it intentionally, it can’t be found. Only the script and the data live in the real world, here the real world being your hard drive. So, the moral of the story is, trust your hard drive more than R’s memory or your own memory and save scripts, not work spaces. If something is very important, save it in multiple places and save multiple versions as you progress so you can always go back.

Now that I have just convinced you to save scripts, how can you be sure your script is saved? Looking at the name of your script in the tab will tell you if it is saved or not. The name of the script will be in black. Now, go to your script and type in the line of code from above setting *z* to 45. The name of your script will change from black to red. This means there are unsaved changes in your script. Make sure you Always save your script to your hard drive, and give it an informative name so you can know what it is in 2 weeks.

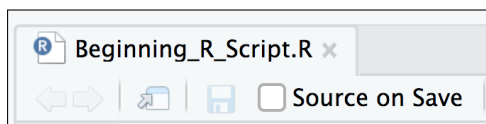


Figure 2.9: Saved Script



Figure 2.10: Unsaved Script

There is one more window you should be familiar with. Go to your script and create a new variable. Assign *A* to some value and run it. Now the name of your script should be shown in red. Now close R Studio. This window is telling you that you have unsaved changes to 2 things: The workspace and the script. Remember, we want to save the script, but we do not want to save the workspace. So here, you can uncheck the Workspace, and just save your script.

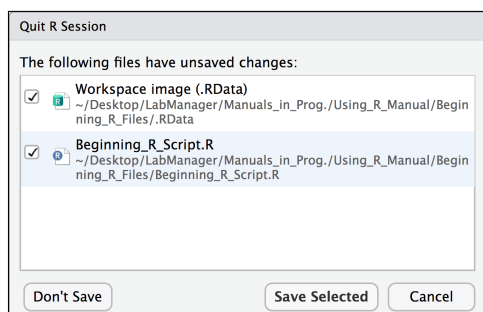


Figure 2.11: Save Session and Script?

Save Scripts, not Workspaces!

3: Using R Markdown Scripts

3.1 Installing R Markdown

Let's get you started with the markdowns because it will be easier to explain why we want to use them once we have one in front of us. Look to the File/Plot/Packages window of R Studio and click on the Packages tab. Once there, you will click on INSTALL in the top left corner of that window shown in 3.1. When you click that the window shown in 3.2 will open. In the Packages field, type "rmarkdown". It should suggest the package for you as you type. Select it and click INSTALL.

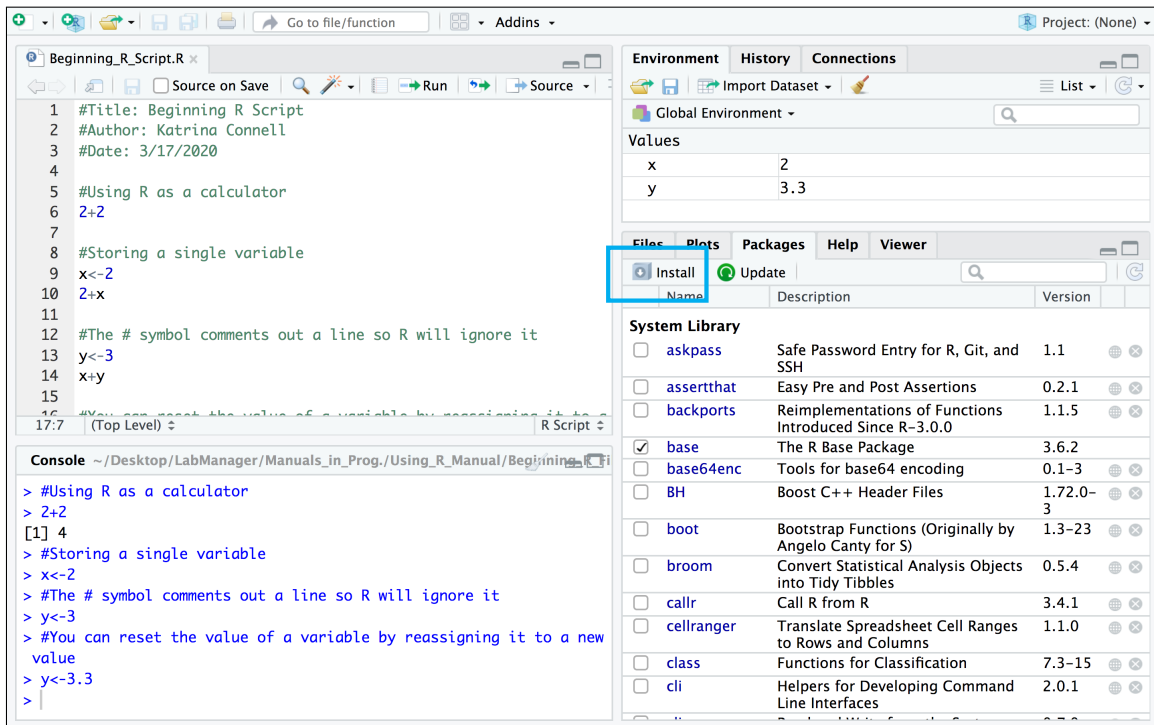


Figure 3.1: Install Packages

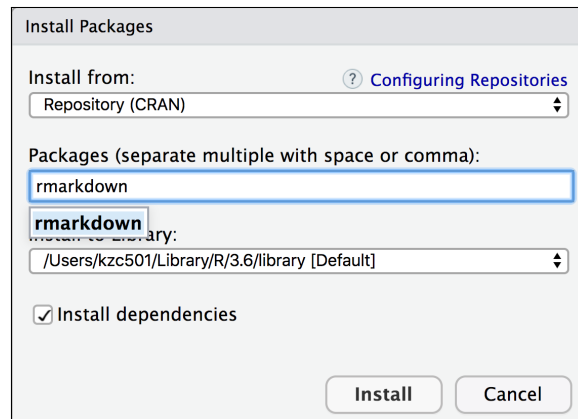


Figure 3.2: Install Packages Window

This should have installed the Rmarkdown package for you. Packages are basically add ons that give Base R more functionality and more power. So, now armed with Rmarkdown, you can open a new type of file, called the R Markdown file as shown in 3.3. This will open up a window (3.4). Here, type the name you want to give this file. For now, click HTML for the Default Formatting Options. We will eventually get to PDFs, but this will require us to install LaTeX (which I highly recommend). Click OK. When the new file opens, it will appear in the Source window, as an R Markdown file is a different type of source script (??).

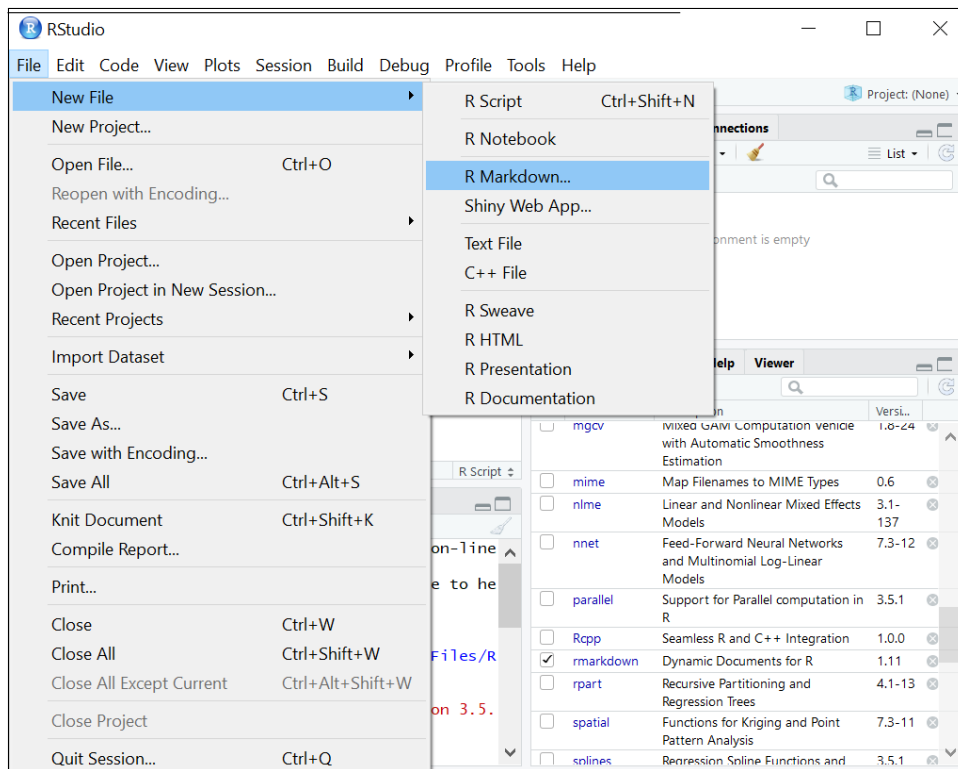


Figure 3.3: Open R New Markdown

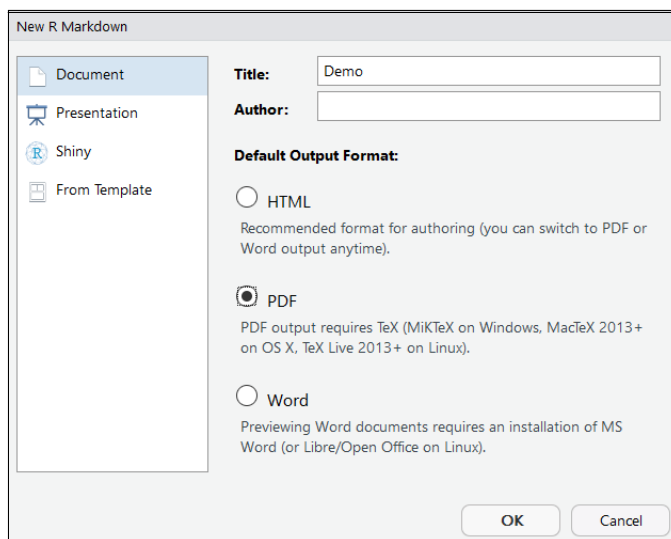


Figure 3.4: Open New R Markdown Window

3.2 Using R Markdown

Go ahead and open the R Markdown file that came with this manual. You can do this by going to FILE->OPEN, or by navigating to it in Finder/File Explorer and double clicking on it. The first thing to do, is to make sure you have you options set for the Markdown. In the top menu, find the gear icon, and click on it. This will give you several options. You can play around with them later, but for the time being, set them how they are shown below.

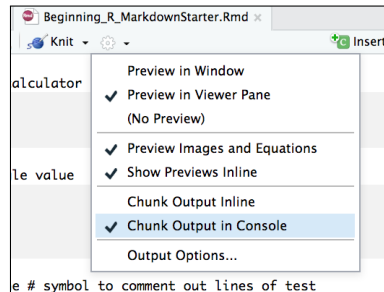


Figure 3.5: Markdown Options

Secondly, there are a few differences in how the Markdown file looks and functions compared to a script.

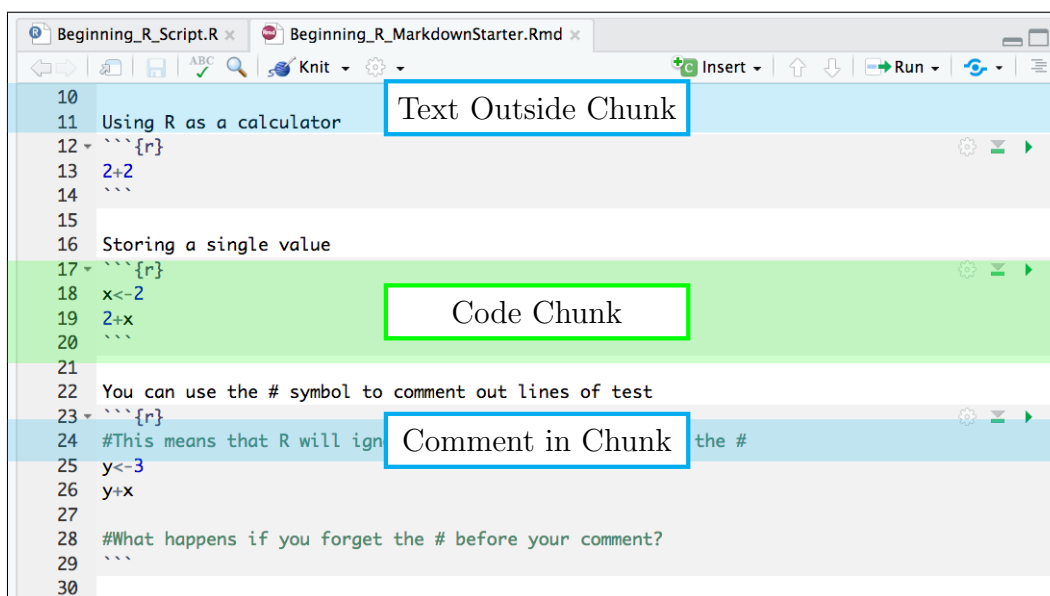


Figure 3.6: R Markdown

The R Markdown file will have three main types of information.

1. Code Chunks - These sections of the markdown are shaded, and being and end with “`````”. The text inside the `{ }` tells R which programming language to use to evaluate that code. Here we are working with R code, so it shows `{r}`. Everything inside of this chunk will be treated as R code.
2. Text Outside of Chunks - Because we have sectioned off the code into it’s chunk, any text outside of the chunk can just be typed in like normal, and does not need the `#` before it. In fact, the `#` has a different function here, that we will come back to.
3. Comments in Chunks - Because R knows that anything inside the chunk is R code, if you want to put text inside of a code chunk, you *will* need to comment it out just like in an R script with the `#` before it.

Each chunk will have 3 icons at the top left corner. These are important ones to know. Later we will come back to the gears icon, so just remember that there is a gear there that will let you set options for the chunks in terms of the warnings and messages shown for that chunk. More on that later. For now, I want you to focus on the other two icons.

```
23 - ```{r}
24 #This means that R will ignore anything that comes after the #
25 y<-3
26 y+x
27
28 #What happens if you forget the # before your comment?
29 ```
```






Figure 3.7: Chunk Options

The first one to know is the Run  button. This button will run all of the lines of code in that chunk. So, instead of placing your cursor and hitting CMD ENTER for each line, you can just click this button, and it will run all of the lines of code in that chunk. Go ahead and click this button for Chunk 2 in the Markdown and see how it runs both lines of code for you.

The other one is the Run All Above  button. This button will run every chunk *above* that one, but not that one itself. This is handy if you change something in the analysis that might affect other pieces of the code and you want to re-run everything to start fresh. Go ahead and scroll down to line 32, and press the Run All Above button to run all of the chunks above that one. Now press the Run button to run that chunk as well.

Here, be sure to **save your Markdown** file. See Chapter 2: Saving Your Work for a refresher on why we save our scripts.

3.2.1 Tidyverse on Linux

[StackExchange Link Here](#)

From Terminal, I ran `sudo apt-get install r-cran-xml r-cran-rcurl`
and then I ran `sudo apt-get install libssl-dev libxml2-dev libcurl4-openssl-dev`

3.2.2 Useful Shortcut

Description	PC	Mac
Insert chunk	Ctrl+Alt+I	Cmd+Option+I
Run all above	Ctrl+Alt+B	Cmd+Option+B
Comment/uncomment	Ctrl+Shift+C	Cmd+Shift+C
Insert assignment operator	Alt+-	Option+-



Practice

At this point, you should walk through the rest of the markdown, reading the text and running each line of code to see what it does.

4: Knitting Markdowns

One additional advantage of the markdown files is that they can create reproducible reports. What I mean by this is we can Knit the file into an HTML document that anyone can open. This file will show the code, and the produced result. This is a wonderful way to be transparent in our research, and to share our analyses with others, especially in collaborative environments.

To Knit the markdown, Find the ball of yarn and knitting needles icon at the top of the source window, as shown in 4.1 and then click KNIT TO HTML. This will open a new Tab in the Console window called R Markdown. This will run each piece of the code and generate a final HTML file. It will open it in the File/Plots/Packages window, and also save it to the Beginning_R_Files folder where the markdown is stored. Go ahead and do this for the markdown you are working with, and either look at the final file in the Viewer window or open up the HTML from the folder.

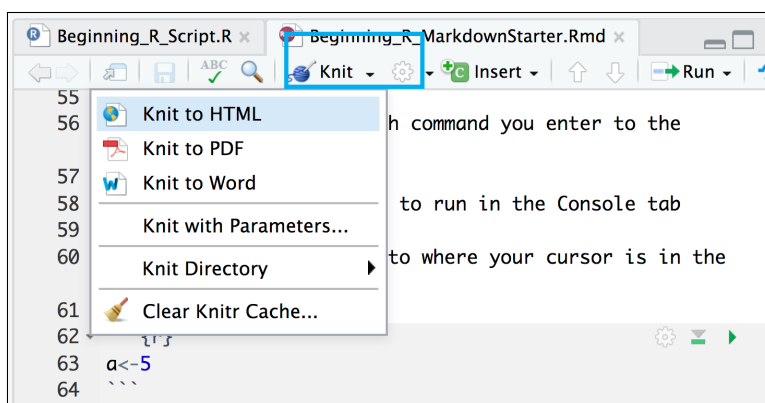


Figure 4.1: Knit

If you have LaTeX installed on your computer, you can Knit to a PDF, which is my preferred format, but requires some additional downloads.

A Few Notes on Knitting

When knitting, R will run every line of code in the markdown. If there are any errors in the markdown, the knitting will not work. To avoid hitting errors when knitting, it is a good idea to select your entire script and run it before knitting, this way you can more easily troubleshoot where any errors might be. Additionally, if you have the command `View(DATA)` in your markdown, it will generate a separate window to view the data. If your data set is large, this is not good, so either comment them out before knitting, or use `head(DATA)` in place of `View`.



Practice

At this point, you should run the Markdown and compare closely what you had in the Source and what that generates in the Markdown.

5: R Projects



BEFORE YOU BEGIN!

Opening up an R project will open up a new **Workspace**. Before you begin, if you have anything else open in R Studio, make sure it is saved To **YOUR HARD DRIVE**, because opening a new R project will open a new workspace, and may close out your old workspace. Anything unsaved will be lost. Refer to Chapter 2: Saving Your Work, specifically, Chapter 2.2: The R Workspace to make sure you have everything saved properly before moving on.

The last piece of Basic R knowledge I would like to share with you is the R Project. At this point in your R career, it may be difficult to understand why this is going to be useful to you, but please just trust me. The R Project is an essential tool for managing your analyses. I can tell you from personal experience and from help many students, graduate and undergraduate, that once you begin to analyses your won data, your analysis will get out of hand *very* quickly if you do not organize your files well from the beginning. You will need to manage potentially multiple markdowns for different versions f the analysis, data files, graphs, knit markdowns, etc. and it will get messy fast. A messy analysis may work at the moment, since everything is fresh in your mind, but if you come back to it even a week later, you will not be able to function nor will you know where anything is. The R Project will be an invaluable tool for you to do this, and so I want you to learn it right from the start and begin your R career with good habits. Before we begin, I want to explain a little as to why these R Projects are a good idea.

5.1 Directories and File Paths

In any computer language, including R, directories are key. The directory is in some ways the folder you are working out of, but it is more than that. The directory, or file path to a directory, is how the computer will look for files on your computer. If you think of your computer as having nested levels, the hard drive itself is the top level. from there, the hard drive is then broken down into smaller sub sections (e.g., Users, Documents, Applications, Desktop etc.) and then each of those sections have sections within them, like your files and folders. In order for the scripting language to find files on your computer, you need to walk it through the whole path for where to find your files from the top. This is your **FILE PATH** that leads to the **DIRECTORY** you are working out of. So, when you go to have R import some data to work with, you would (usually) supply it with the full file path to where the data set is located on your computer. If we were trying to tell R where to find a data set in our “data” folder, we could use the following Absolute path.

Mac - /Users/USERNAME/Desktop/Using_R_Manual/Beginning_R.Files/data/dataset1

PC - C:\USERNAME\Desktop\Using_R_Manual\Beginning_R.Files\data\dataset1

Regardless of Mac vs. PC, the file paths work more or less the same. In this case we are telling the program to start at the hard drive, and work it’s way down the provided file path into the correct user account, then to the desktop, then inside each folder until we arrive at *dataset1*. This is shown in Figure 5.1. This is an absolute file path, where we have specified every step the program must take to find the file, and the start of the path is always the hard drive as a whole. This is perfectly fine, however; if you change **ANYTHING** in the structure above that file, the path is broken, and the program will not be able to find your files. If, for example, you move the *Using_R_Manual* folder to Documents, or change it’s name, that absolute path will no longer work. Also, if you send your analysis to someone else, clearly their computer will not have the same user name as you, and so the path will not work.

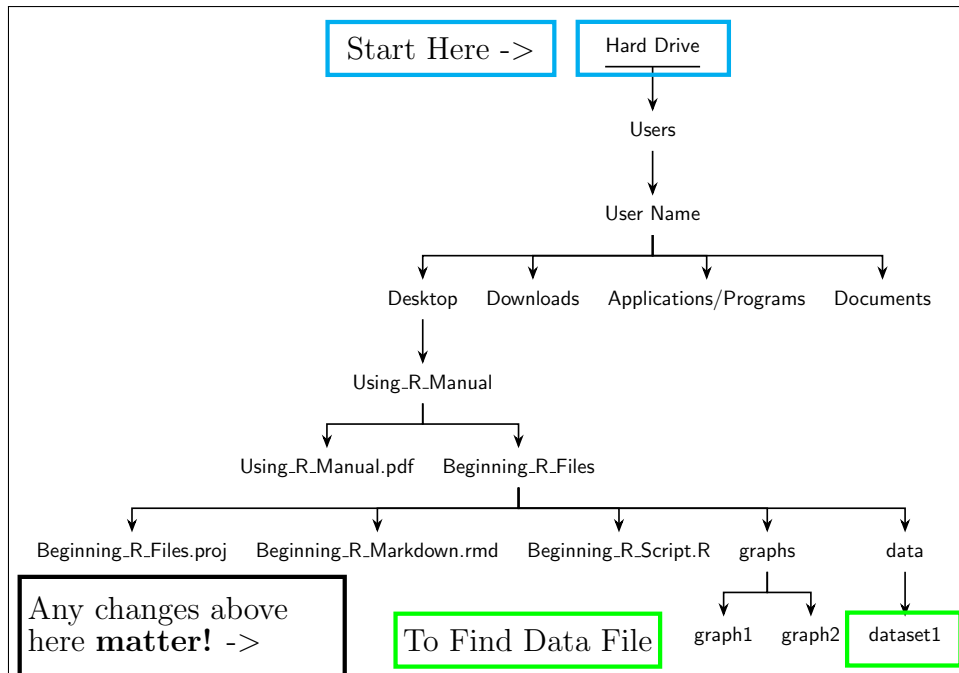


Figure 5.1: Absolute File Path Visual

This is why a relative path is ideal, and this is what the R Projects set up for you automatically. The R Project sets up a relative path to the folder *where the R Project is created and lives*. This means, that the program will automatically find the path above that folder shown in Figure 5.2, and the only path you have to set will be any deviations from that relative path. So, in this case, with a relative path set to the Beginning_R_Files folder, all we need to do is specify that our data is store in a folder *inside* of the relative path folder. This would give us the paths below. The dot slash is how we tell R that the path is relative, and then we say ” OK, you already know to go to the Beginning_R_Files folder, but go one deeper into the “data” folder to find this file”.

Mac - `./data/dataset1`

PC - `..\data\dataset1`

By doing this, R is able to find our data no matter what we do to the path above the Beginning_R_Files folder, or on any computer. ANY change above that folder doesn’t matter, the path will still work. This means you could change file names, move folders, put it on someone else computer, and it will still work. This saves you and anyone you send this analysis, time and effort trying to fix all of the file paths.

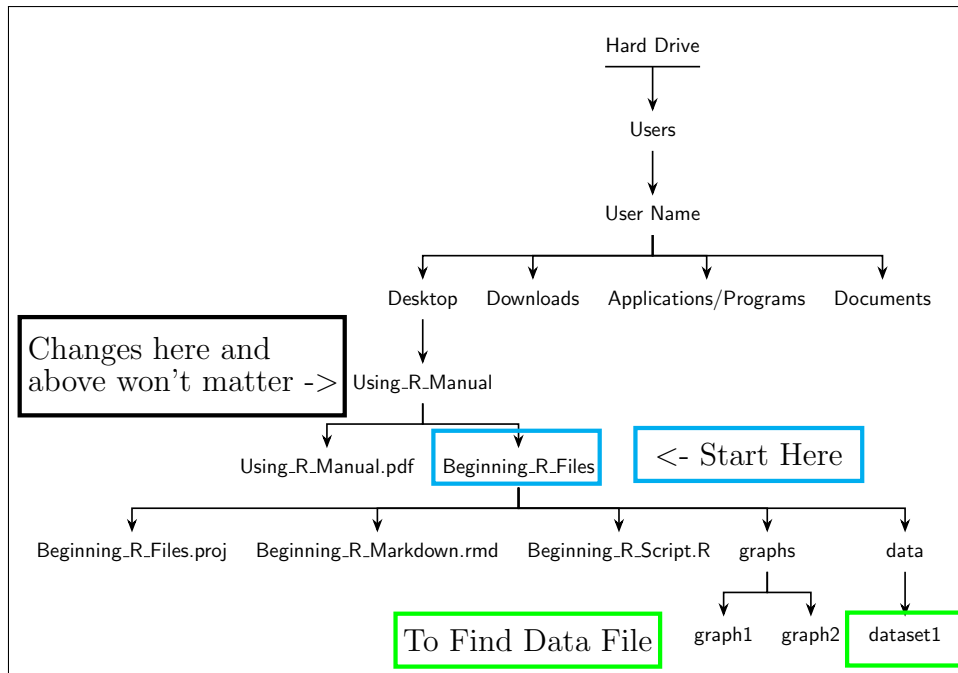


Figure 5.2: Relative File Path Visual

5.2 Using R Projects

BEFORE YOU BEGIN!

Save your work! Anything unsaved could be lost when you open a new R Project. Refer to Chapter 2: Saving Your Work, specifically, Chapter 2.2: The R Workspace to make sure you have everything saved properly before moving on.

So, hopefully that was enough to convince you to use the R Projects, and now you want to create one. To do this, go to FILE->NEW R PROJECT shown in 5.3.

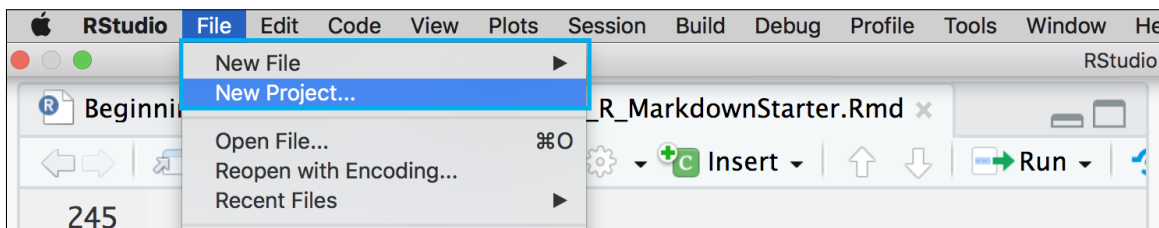


Figure 5.3: New R Project

The first window that you will see will ask you if you want to create a new directory or use an existing one. If you are just starting a brand new analysis, go ahead and select New Directory. Chances are though, that you have already begun working on something, and will want to use an existing directory (folder). Right now, you already have the folder you where the scripts we have been working with were saved, so you can go ahead and select Existing Directory 5.4.

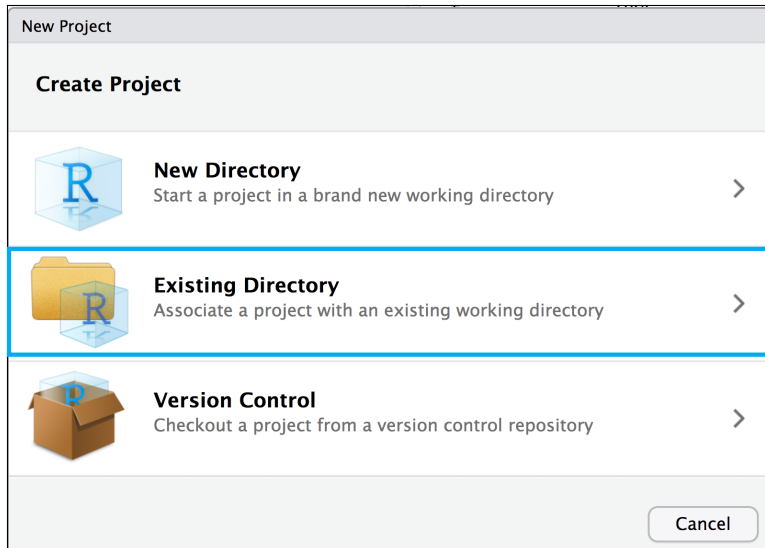


Figure 5.4: New R Project Step 1

This will bring up another window that will ask you which directory (folder) you would like to use. It should default to the folder where your scripts were saved, but if not, click browse and find the folder you want. Click CREATE PROJECT when finished 5.5.

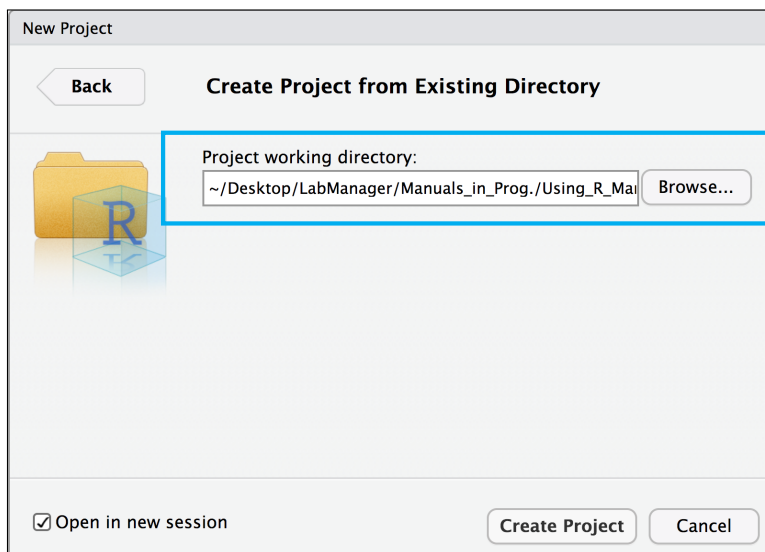


Figure 5.5: New R Project Step 2

You have now officially created your first R Project! Welcome to the wonderful world of organized analyses. Figure 5.6 shows you what your R Studio session should look like now. Everything will look the same, but there are a few nice features that will make your life easier.

First, look to the Files/Plots/Packages window and click on Files if it isn't already selected. Here, you will now see a breakdown of all the files in your directory, including sub directories of the folders. You can now open the markdown files from here. Go ahead and click on it to open it.

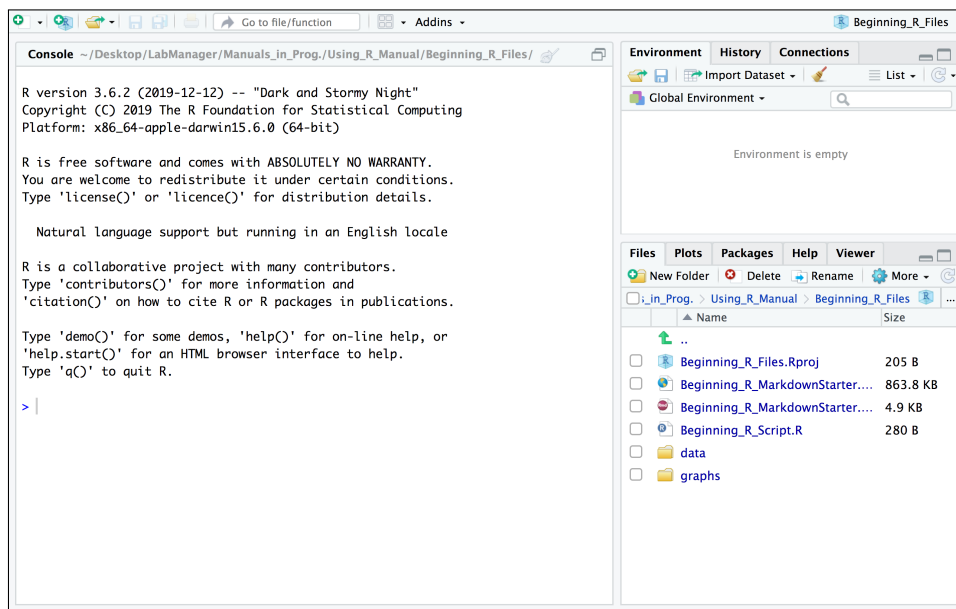


Figure 5.6: New R Project Step 3

Thanks for reading!

I know learning R can seem daunting, so I hope this manual provides you with some of the basics you need to get started in R. It's worth the effort, trust me :)

Good Luck and Happy Coding!
Katrina